A6: SORTING AND

"Chaos was the law of nature; Order was the dream of man." – Henry Adams

Course: CS 5006
Summer 2018
Due: 22 June 2018, 5pm

OBJECTIVES

After you complete this assignment, you will be comfortable with:

- Sorting!
- Quicksort
- Heap sort
- Priority Queues
- Linear-time sorting

RELEVANT READING

- Kleinberg and Tardos, Chapter …

AND NOW ON TO THE FUN STUFF.

Problem 1: 1 (3 points)
Illustrate the operation of INSERTION-SORT on the array [31; 41; 59; 26; 41; 58]

[31; 41; 59; 26; 41; 58]; assume array index starts at 1.
for j = 2 to A:length; key = A[2] = 41; i = j-1 =1; 1>0 but 31 < 41; A[1+1] = 41;
for j = 3, key = A[3] = 59; i = j-1 =2; 2>0 but 41 < 59; A[2+1] = 59;
for j = 4, key = A[4] = 26; i = j-1 =3; 3>0 and 59 > 26;

[31; 41; 59; 59; 41; 58]; i = 3-1 = 2; 2>0 and 41 > 26;
[31; 41; 41; 59; 41; 58]; i = 2-1 = 1; 1>0 and 31>26;
[31; 31; 41; 59; 41; 58]; i = 1-1 = 0; 0$\not>$0;
[26; 31; 41; 59; 41; 58];

for j = 5; key = 41; i = 5-1 = 4; 4 > 0 and 59>41;
[26; 31; 41; 59; 59; 58]; I = 4-1 = 3; 3>0 but 41 = 41;
[26; 31; 41; 41; 59; 58];

For j = 6; key = 58; I = 6-1 = 5; 5>0 and 59 > 58;
[26; 31; 41; 41; 59; 59]; i = 5-1 = 4; 4>0, but 41 $\not>$ 58;
[26; 31; 41; 41; 58; 59]

Problem 2: Correctness of Bubblesort (4 points)

Bubblesort is a popular, but inefficient, sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

```
1     BUBBLESORT(A)
2    for i = 1 to A:length
3     for j = A:length down to i +1
4        if A[ j] < A[ j -1]
5           exchange  A[ j] with A[ j- 1]
```

(a) (2 points) Let A_t denote the output of BUBBLESORT(A). To prove that BUBBLESORT is correct, we need to prove that it terminates and that $A_t[1] \leq A_t[2] \leq \ldots \leq A_t[n]$ where $n = A.length$. In order to show that BUBBLESORT actually sorts, what else do we need to prove?

We need to prove that every item in the array is being compared or considered at least once.

(b) (2 points) What is the worst-case running time of bubblesort? How does it compare to the running time of insertion sort?

Worse-case running time for bubblesort is n^2.

For insertion sort, the worse case runtime is n^2 bubblesort's running time is the same as insertion sort.

Problem 3: 3 (2 points)

What are the minimum and maximum number of elements in a heap of height h?

Minimum number of elements in a heap of height h is $2^h - (2^{h-2} + 1)$ to account for all situation were there may only be one leaf in the final level assignment to each node in the level above it.

The maximum number of elements in a heap of height h is $2^h - 1$ assuming the heap is each node in the level before the last has 2 children.

Problem 4: 4 (4 points)

Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree.

Assume the root of the subtree isn't the the largest value occurring anywhere in the subtree. For something to be a max-heap, by definition, the value of each node is less than or equal to the value of it's parent. If you have a subtree, where the value of the root is less than the value of any of the children or descendent nodes, it can't be a max-heap. Therefore, the root of any subtree in max-heap contains the largest value occurring anywhere in that subtree.

Problem 5: 5 (2 points)

Is an array that is in sorted (ascending) order a min-heap?

If set up with these requirements: left child of k sits in position 2k of the array and the right child in 2k+1, and the parent is in position k/2, yes. Otherwise, without that information, no.

Problem 6: 6 (2 points)

Is the array with values [23; 17; 14; 6; 13; 10; 1; 5; 7; 12] a max-heap? Provide some explanation.

Yes. Assuming the array is set up with these requirements: left child of k sits in position 2k and the right child in 2k+1, and the parent is in position k/2. As you go down the array, putting things in order, eventually you get to a point where you have to place the 12 somewhere. Assuming that the above requirements hold, the parent of 12 is in position 5, which has the value of 13. By definition a max-heap requires each node to be less than or equal to the value of it's parent, with the maximum-value element at the root. 12 is smaller than 23, which is the root value, so that provision is met and 12's parent, 13 is higher than 12, so that provision is met, so this is a max-heap.

Problem 7: 7 (4 points)

Illustrate the operation of HEAPSORT on the array A:[5; 13; 2; 25; 7; 17; 20; 8; 4].

Assuming this is a built heap:

i = 8; swap(5, 4); [4; 13; 2; 25; 7; 17; 20; 8; 5]; n = 8-1 = 7; Heapify(A, 1);
[13; 4; 2; 25; 7; 17; 20; 8; 5], [13; 25; 2; 4; 7; 17; 20; 84], [13; 25; 2; 84; 4; 7; 17; 20];
i = 7; swap (13, 17); [17, 25, 2, 84, 4, 7, 13, 20]; n = 7-1 = 6; Heapify(A, 1);
[25, 17, 2, 84, 4, 7, 13, 20]; [25, 84, 2, 17, 4, 7, 13, 20]; [25, 84, 2, 20, 4, 7, 13, 17];
i = 6; swap(25, 7); [7, 84, 2, 20, 4, 25, 13, 17];n = 6-1 = 5; Heapify(A, 1)[
[84, 7, 2, 20, 4, 25, 13, 17]; [84, 20, 2, 7, 4, 25, 13, 17]; [84, 20, 2, 17, 4, 25, 13, 7];
I = 5; swap(84, 4); [4, 20, 2, 17, 84, 25, 13, 7]; n = 5-1 = 4; Heapify(A, 1);
[20, 4, 2, 17, 84, 25, 13, 7]; [20, 84, 2, 17, 4, 25, 13, 7];
i = 4; Swap(20, 17); [17, 84, 2, 20, 4, 25, 13, 7]; i = 4-1 = 3; Heapify(A, 1);
[84, 17, 2, 20, 4, 25, 13, 7]; [84, 20, 2, 17, 4, 25, 13, 7];
i = 3; swap(84, 2); [2, 20, 84, 17, 4, 25, 13, 7]; i = 2; Heapify(A, 1);
[84, 20, 2, 17, 4, 25, 13, 7]; i = 2; swap(84, 20) = [20, 84, 2, 17, 4, 25, 13, 7]; i = 1; Heapify(A, 1);
[84, 20, 2, 17, 4, 25, 13, 7]

Problem 8: 8 (2 points)

Illustrate the operation of PARTITION on the array A: [13; 19; 9; 5; 12; 8; 7; 4; 21; 2; 6; 11]. Assume the pivot is the first element. (You can show just the final output, not all the intermediate steps).

[9; 5; 12; 8; 7; 4; 2; 6; 11; 13; 19; 21]

Problem 9: 9 (2

What is the running time of QUICKSORT when all elements of array A have the same value?

n^2 because you can't split the array into smaller problems since everything is the same. Nothing will get rearrange each time and you are deal with the same size problem each time.

Problem 10: 10 (3 points)

We talked about randomizing QUICKSORT. Why do we analyze the expected running time of a randomized algorithm and not its worst-case running time?

When we started talking about it in class, we accepted that in the worse case, quicksort was had a of $\Theta(n^2)$. But we recognized that quicksort is supposed to the quickest sort by a multiplicative constant factor. This can only be achieved with randomization. Furthermore, we typically will never see the worst-case scenario because quicksort is implemented with randomization built in because of that worse-case scenario. So for all effects and purposes, the worst case scenario is a randomized quicksort.

Problem 11: 11 (3 points)

Illustrate the operation of COUNTING-SORT on the array A: [6; 0; 2; 0; 1; 3; 4; 6; 1; 3; 2].

[6; 0; 2; 0; 1; 3; 4; 6; 1; 3; 2]; new array counts [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];
2 0's, 2 1's, 2 2's, 2 3's, 1 4, 2 6's; counts [2; 2; 2; 2; 1; 0; 2];
counts[i] = counts[1] + counts[i-1]; counts[2; 4; 6; 8; 9; 9; 11];
2
use value output array[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0]; counts[2; 4; 5; 8; 9; 9; 11];
output array [0, 0, 0, 0, 0, 2, 0, 3, 0, 0, 0]; counts[2; 4; 5; 7; 9; 9; 11];
output array [0, 0, 0, 1, 0, 2, 0, 3, 0, 0, 0]; counts[2; 3; 5; 7; 9; 9; 11];
output array [0, 0, 0, 1, 0, 2, 0, 3, 0, 0, 6]; counts[2; 3; 5; 7; 9; 9; 10];
output array [0, 0, 0, 1, 0, 2, 0, 3, 4, 0, 6]; counts[2; 3; 5; 7; 8; 9; 10];
output array [0, 0, 0, 1, 0, 2, 3, 3, 4, 0, 6]; counts[2; 3; 5; 6; 8; 9; 10];
output array [0, 0, 1, 1, 0, 2, 3, 3, 4, 0, 6]; counts[2; 2; 5; 6; 8; 9; 10];

Problem 12: 12 (3

Illustrate the operation of RADIX-SORT on the following list of English words: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

Problem 13: 13 (3 points)

Illustrate the operation of BUCKET-SORT on the array A:[79; :13; :16; :64; :39; :20; :89; :53; :71; :42].

Problem 14: 14 (3

Explain why the worst-case running time for BUCKET-SORT is $\Theta(n^2)$ What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \lg n)$? (Assume the sort used in each bucket is INSERTION-SORT)

SUBMISSION DETAILS

Things to submit:

- Submit your assignment in your Github repo.
- The written parts of this assignment as a .pdf named "CS5006_[lastname]_A6.pdf". For example, my file would be named "CS5006_Slaughter_A6.pdf". (There should be no brackets around your name).
- Make sure your name is in the document as well (e.g., written on the top of the first page).
- Make sure your assignment is in the A5 folder in your Github repo.

HELPFUL HINTS

- Ask clarification questions on Piazza.
- Remember, your write-up should convince graders and instructors that you are providing your own work and should showcase your understanding.
- Use the resources page on the course website for supplemental materials.
- In general, problems will be graded both on whether you are taking the right approach and whether you got the right answer. So, show your work and explain your thinking.